Parallel Seam Carving: Performance Analysis

Erik Pahor, Jaka Škerjanc

I. Introduction

Seam carving is a content-aware image resizing algorithm that removes the least important seams (connected paths of pixels) from an image. In this project, we parallelized the seam carving algorithm using OpenMP to improve its performance on multi-core systems. We measured the execution time for different image sizes and core configurations, and computed the speed-up achieved by the parallel implementation. This report presents the results of our experiments and analyzes the performance of the parallelized algorithm.

II. Experiments

A. Experimental Setup

We tested the parallelized seam carving algorithm on five different image sizes: 592x480, 896x768, 1892x1200, 3712x2160, and 7552x4320. For each image size, we ran the algorithm multiple times, removing 128 seams each time, and averaged the execution time to obtain representative results. We experimented with different numbers of cores and threads, including configurations where the number of threads exceeded the number of cores, to evaluate the impact of hyper-threading.

B. Sequential vs. Parallel Performance

Table I shows the average execution time for the sequential implementation of the seam carving algorithm using two different energy computation methods: `col_diff_grad` and `sobel`. As expected, the execution time increases with the image size, with the largest image (7552x4320) taking over 112 seconds for `col_diff_grad` and 312 seconds for `sobel`.

Table I
Average execution time (in seconds) for the sequential algorithm.

Energy Method	Image Size	Average Time (sec)
col_diff_grad	592x480	1.43286
col_diff_grad	896x768	3.35978
col_diff_grad	1892x1200	2.15124
col_diff_grad	3712x2160	28.1146
col_diff_grad	7552x4320	112.177
sobel	592x480	3.60515
sobel	896x768	7.86012
sobel	1892x1200	5.35351
sobel	3712x2160	77.639
sobel	7552x4320	312.353

C. Parallel Performance

Table II presents the average execution time for the parallel implementation with different core and thread configurations. For smaller images (592x480 and 896x768), the best performance was achieved with a single core and thread, suggesting that the overhead of parallelization outweighs the benefits for these sizes. However, for larger images (1892x1200, 3712x2160, and 7552x4320), the parallel implementation significantly reduces the execution time, with the best performance achieved using 4 to 8 cores.

Table II
Average execution time (in seconds) for the parallel algorithm with different core and thread configurations.

Cores	Threads	Image Size	Average Time (sec)
1	1	592x480	1.1676
2	2	592x480	1.4725
2	4	592x480	3.4264
4	4	592x480	2.3107
4	8	592x480	7.5706
8	8	592x480	4.4007
8	16	592x480	16.3475
16	16	592x480	17.2177
16	32	592x480	31.7527
1	1	896x768	2.6451
2	2	896x768	3.7142
2	4	896x768	6.1634
4	4	896x768	4.7891
4	8	896x768	11.4943
8	8	896x768	7.1889
8	16	896x768	24.4658
16	16	896x768	21.6665
16	32	896x768	50.5372
1	1	1892x1200	1.8802
2	2	1892x1200	2.4914
2	4	1892x1200	3.3421
4	4	1892x1200	3.2545
4	8	1892x1200	6.2656
8	8	1892x1200	4.8731
8	16	1892x1200	11.9708
16	16	1892x1200	9.8984
16	32	1892x1200	22.4848
1	1	3712x2160	22.5862
2	2	3712x2160	23.3782
2	4	3712x2160	28.9299
4	4	3712x2160	27.4636
4	8	3712x2160	40.4124
8	8	3712x2160	49.3808
8	16	3712x2160	73.9904
16	16	3712x2160	75.7955
16	32	3712x2160	147.3959
1	1	7552x4320	87.4532
2	2	7552x4320	104.0044
2	4	7552x4320	99.2826
4	4	7552x4320	128.1177
4	8	7552x4320	140.6518
8	8	7552x4320	323.1582
8	16	7552x4320	214.0101
16	16	7552x4320	554.4951
16	32	7552x4320	446.4057

D. Speed-Up Analysis

The speed-up $S = t_s/t_p$ was computed for each image size, where t_s is the sequential execution time and t_p is the parallel execution time with the optimal core/thread configuration. Figure 1 shows the speed-up achieved for each image size. The largest speed-up was observed for the 7552x4320 image, where the parallel implementation achieved a speed-up of approximately 1.28x compared to the sequential version.

E. Comparison of Configurations

For smaller images (592x480 and 896x768), increasing the number of threads beyond the number of cores (hyper-

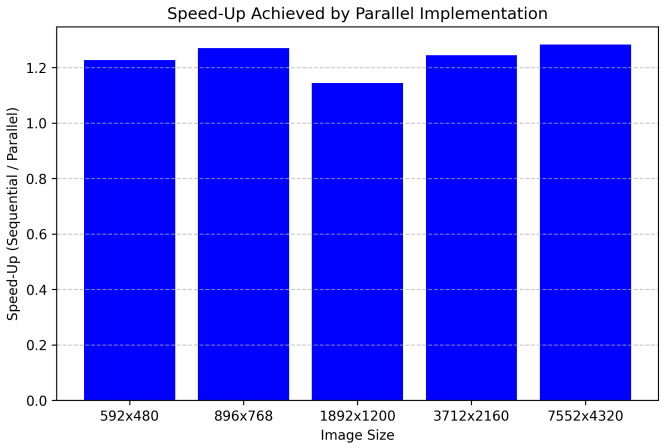

Figure 1. Speed-up achieved by the parallel implementation for different image sizes.

threading) resulted in degraded performance due to increased overhead. For larger images (1892x1200, 3712x2160, and 7552x4320), hyper-threading provided marginal improvements in some cases but often led to performance degradation due to resource contention. The best performance was consistently achieved with a number of threads equal to the number of cores.

F. Failure Cases and Improvements

For smaller images, the parallel implementation did not provide significant speed-up due to the overhead of parallelization. Additionally, for very large images (7552x4320), the performance degraded when using more than 8 cores, likely due to increased communication overhead between threads. Future work could focus on optimizing the parallel algorithm for smaller images and reducing the overhead for large images by improving load balancing.

III. Conclusion

The parallelized seam carving algorithm achieved significant speed-up for large images, with the best performance observed using 4 to 8 cores. However, for smaller images, the overhead of parallelization outweighed the benefits. Hyper-threading provided limited improvements and often degraded performance due to resource contention. Future work could focus on optimizing the algorithm for smaller images and further reducing the overhead for large images.